

ISE

Industrial and
Systems Engineering

Optimal Generalized Decision Trees via Integer Programming

MATT MENICKELLY

Department of Industrial and Systems Engineering, Lehigh University, Harold S. Mohler
Laboratory, 200 West Packer Avenue, Bethlehem, PA 18015-1582, USA.

OKTAY GÜNLÜK

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.

JAYANT KALAGNANAM

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.

KATYA SCHEINBERG

Department of Industrial and Systems Engineering, Lehigh University, Harold S. Mohler
Laboratory, 200 West Packer Avenue, Bethlehem, PA 18015-1582, USA.

COR@L Technical Report 13T-02-R1



LEHIGH
UNIVERSITY.

COR@L
COMPUTATIONAL OPTIMIZATION
RESEARCH AT LEHIGH



Optimal Generalized Decision Trees via Integer Programming

MATT MENICKELLY^{*1}, OKTAY GÜNLÜK^{†2}, JAYANT KALAGNANAM^{‡3},
AND KATYA SCHEINBERG^{§4}

¹Department of Industrial and Systems Engineering, Lehigh University, Harold S. Mohler Laboratory, 200 West Packer Avenue, Bethlehem, PA 18015-1582, USA.

²IBM T.J. Watson Research Center, Yorktown Height, NY 10598, USA.

³IBM T.J. Watson Research Center, Yorktown Height, NY 10598, USA.

⁴Department of Industrial and Systems Engineering, Lehigh University, Harold S. Mohler Laboratory, 200 West Packer Avenue, Bethlehem, PA 18015-1582, USA.

December 19, 2016

Abstract

Decision trees have been a very popular class of predictive models for decades due to their interpretability and good performance on categorical features. However, they are not always robust and tend to overfit the data. Additionally, if allowed to grow large, they lose interpretability. In this paper, we present a novel mixed integer programming formulation to construct optimal decision trees of specified size. We take special structure of categorical features into account and allow combinatorial decisions (based on subsets of values of such a feature) at each node. We show that very good accuracy can be achieved with small trees using moderately-sized training sets. The optimization problems we solve are easily tractable with modern solvers.

1 Introduction

Decision trees (DTs, for short) are a class of predictive models that predict an output label for given input data based on a sequence of binary “tests” or *decisions*. There has been a large amount of literature on DTs since the 1980’s (see the recent survey [6] and references therein). DTs are one of the most popular tools of machine learning and data analysis due to several important advantages they hold over other predictive models. One such advantage is their easy interpretability as the connection between the input and the resulting output is transparent. Moreover, they often naturally result in feature selection, since only a part of the input is typically used in the decisions. Another advantage is that DTs can work with both numerical and categorical data directly. Most other popular classifiers, such as linear classifiers or neural networks, assume the data is numerical.

^{*}mjm412@lehigh.edu. The work of this author is partially supported by NSF Grant DMS 13-19356.

[†]gunluk@us.ibm.com.

[‡]jayant@us.ibm.com.

[§]katyas@lehigh.edu. The work of this author is partially supported by NSF Grants, DMS 13-19356 and CCF-1320137 and AFOSR Grant FA9550-11-1-0239.

Moreover, these classifiers treat the numerical features as *unbounded* continuous quantities, regardless of the actual range of values these features can take. In particular, a categorical feature that can take three values is often represented by a group of three binary variables, so that only one of the variables is equal to 1 and the other two are 0. Then, a linear classifier or a neural network (let us call them “numerical classifiers”) will treat the group of three variables as three continuous variables, which can take any combination of values - essentially ignoring the valuable information that only three values of the triple are possible. The numerical classifiers hope to recover this lost information by observing enough data and fitting the model accordingly. However, it is not a trivial task, especially for linear classifiers, and may require a more complex model than what is really necessary. On the other hand, DTs can be viewed as “logical” classifiers, where one can explicitly use the fact that variables can only take binary values. Indeed it has been established in practice that DTs are most effective for integer and categorical data.

There are also disadvantages to the known DT predictors; in particular, they are not always robust with respect to the training data and they might result in poor prediction on out-of-sample data. The poor prediction is often a result of overfitting and this usually happens when the tree is too large. Hence, small trees are often desirable to avoid overfitting and also for the sake of interpretability. Assuming that for a given data distribution there exists a small DT that can achieve good accuracy, the small DTs that are computed by a typical DT algorithm (such as CART [3, 10]) may not achieve such accuracy, due to the heuristic nature of the algorithms. Moreover, it is usually impossible to establish a bound on the difference of expected loss of the DT produced by a heuristic algorithm and an optimal DT. Another weakness of most known DTs is that decisions that involve categorical features are only based on whether or not that feature takes one particular value. For example, if a categorical feature represents a person’s marital status and can take the values “single”, “married”, “divorced”, “separated”, “widowed”, or “has domestic partner”, a typical DT will make decisions based on a feature being “single” or not, while a more appropriate decision may be “either single or divorced” or not.

In this paper, we aim to find *optimal* small DTs for binary classification problems that produce *interpretable* and *accurate* classifiers for the data for which such classifiers exist. We allow complex branching rules using subsets of values of categorical features (e.g. “either single or divorced” or not), rather than only one value (“single” or not); hence, our small trees are more powerful than the standard DTs. Note that current DT algorithms do not naturally extend to such generalized branching. For example, consider a categorical variable that can take ℓ values. Then, there are $2^{(\ell-1)}$ possible subsets of values of this feature that can be used for branching - a greedy method of choosing a branching rule, such as those used in CART, has to consider all such subsets.

While finding an optimal DT (even without the generalized decisions) is known to be an NP-hard problem, that does not mean that it is always impractical to try to solve this problem using the modern techniques that have gone way beyond simple Branch & Bound (B & B) enumeration [7]. Thus, in this paper, we propose a global optimization approach for classification DTs, which is based on a novel mixed integer linear programming (MILP) formulation of the optimal DT problem. As such, we can utilize significant advances in the power of modern MILP solvers [2, 1]. Moreover, since we directly optimize the empirical loss of a DT, even suboptimal feasible solutions tend to yield classifiers that outperform those learned by other DT algorithms. While we do not propose any particular algorithm for solving the formulation, we show that by using a state-of-the-art MILP B&B-based solver, we can obtain optimal or nearly optimal trees of small depth for a variety of standard data sets.

It is well known in the optimization community that the true computational difficulty of an NP-hard problem can vary tremendously depending on its MILP formulation. Different formulations of the same problem can result in many orders of magnitude speed-up in solution time when a B&B method is applied. The main idea in this paper is to utilize the particular structure of the class of problems for which DTs are deemed most suitable to produce a tractable MILP formulation. In particular, we consider a binary classification problem, which means that the output nodes (leaves) of our DTs generate binary output. Our problem formulation takes particular advantage of this fact. Also, while our formulation can be generalized to continuous data, it is designed for the case when the input data is binary. Hence, for most of the paper, we will consider input data as being a binary vector with the property that features are grouped so that only one feature can take the value 1 in each group for each data sample. Our formulation explicitly takes this structure into account, while allowing decisions based on a subset of features from one group. In this paper we focus on constructing small DTs with up to three levels of decisions, which makes the resulting model clearly interpretable and easily usable by humans. The MILP formulation, in principle, can work for binary trees of any depth, but the depth has to be decided a priori. Our computational results show that the small trees we construct have much better prediction accuracy than popular heuristic methods such as C4.5 [10] and random forests [4]. Extensions to larger trees and larger data sets are a subject of future research.

The rest of the paper is organized as follows: In the next section we describe the main ideas of our approach and the structure of the data for which the model is developed. In Section 3 we describe an initial MILP model and several techniques for strengthening this formulation. We present some computational studies and comparisons in Section 4. Finally, we suggest some possible extensions in Section 5.

2 Setting

For the majority of the paper, we will consider datasets $\{(a^i, y^i) : i \in 1, 2, \dots, N\}$ where $a^i \in \{0, 1\}^d$ is a binary vector, and $y^i \in \{-1, +1\}$ is the class label associated with a negative or positive class, respectively. Additionally, assume that the set of features $\{1, \dots, d\}$ can be partitioned into G nonoverlapping groups, $\{g_1, g_2, \dots, g_G\}$, so that for all our data samples (a, y) , the binary subvector a_{g_j} contains one and only one 1, for each group g_j . For example, let a represent a census data with G categorical features. One such feature, say the j -th, may represent a person’s occupation/employment type out of a list of ℓ_j possible types (e.g., as in the *adult* data set from UCI Machine Learning Repository [8], “Tech-support”, “Craft-repair”, “Other-service”, “Sales”, “Exec-managerial”, “Prof-specialty”, “Handlers-cleaners”, etc.). Then a_{g_j} is a binary subvector with 1 indicating which of the ℓ_j occupation types apply to a given individual.

We note that any typical form of data can be represented by the binary vectors as described. For example, not only a categorical, but also an integer feature that takes ℓ possible values can be, and often is, represented as a set of ℓ binary features, where 1 indicates which value the original feature takes and the rest are 0. Also, any feature that is originally binary can be represented by a group of two binary features - itself and its complement, hence defining a group with one and only one 1. Finally, a real-valued feature can be, when appropriate, made into a categorical one by “bucketing” - that is breaking up the range of that feature into segments and considering segment membership as a categorical feature. This is not uncommon with some data sets; for example, while the age of a person is a continuous variable, it is often more robust, say for advertising purposes, to

Figure 1: Three possible tree topologies: depth 2, depth 2.5 and depth 3.

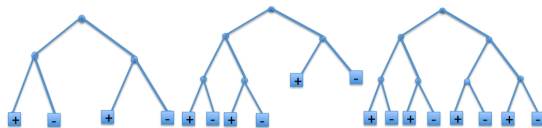
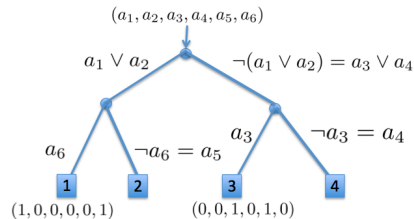


Figure 2: A decision tree example



represent individual by age groups such as “teens”, “young adults”, “middle aged” and “seniors”. For the real-valued data that should not be bucketed, DTs may not be the best choice of classifiers. However, later in the paper, we will extend our model to handle these real-valued variables directly.

Let us now consider the DT models we aim to learn from our data. First, we select a fixed binary DT structure like the ones in Figure 1. The structure consists of *decision nodes* - intermediate nodes and the root - and the prediction nodes - the *leaves*. Each decision node corresponds to a branching rule that has to take the following form: given a set of features that all belong to the same group, branch to the left if the set of features contains a 1 (for a given data sample), otherwise branch to the right. For example, a branching rule for the sensus data could read as follows: if a person’s occupation type is “Sales” or “Exec-managerial” or “Prof-specialty”, then branch to the left, otherwise branch to the right. Hence, the branching rules for each decision node that need to be learned are the group and the particular set of features within the group which determine the direction of branching.

Each leaf corresponds to either a positive or a negative label, meaning that each sample is assigned the label of the leaf where it ends up. Notice that the assignment of the labels to the leaves is fixed, in an alternating manner, and is not learned. This is done without loss of generality, since the concepts of “left” and “right” in the tree are interchangeable. The a priori choice of the tree topology is also without loss of generality, in the following sense: by learning an optimal tree of a given topology we automatically optimize over all trees whose topology is a subtree of the chosen topology. This is simply because some leaves may correspond to an empty set of samples, so that the branching decision at the preceding decision node is redundant. However, fixing a tree with a simple topology results in an easier problem, and is hence desirable. In our computational results, we will explore trees with the three topologies shown in Figure 1.

To give a concrete example, let us consider a tree in Figure 2 applied to binary vectors $(a_1, a_2, a_3, a_4, a_5, a_6)$ consisting of two binary groups: $\{a_1, a_2, a_3, a_4\}$ and $\{a_5, a_6\}$. The branching decision at the root, is based on whether a_1 or a_2 are equal to 1. If true, a given data sample is routed to the left, otherwise (that is if both a_1 and a_2 are 0 and, hence either $a_3 = 1$ or $a_4 = 1$), the sample is routed to the right. The branching at the other two decision nodes are analogous and are shown in the picture. We can now see that data sample $a^1 = (1, 0, 0, 0, 0, 1)$ is routed to leaf 1 and data sample $a^2 = (0, 0, 1, 0, 1, 0)$ is routed to leaf 3.

Our class of feasible trees, thus, is the set of trees of a given topology with any assignment of groups and corresponding sets of features to each of the decision nodes. The ultimate goal is to find the optimal assignment that maximizes the number of *correct classifications* of all samples in the training set. The classification of the i th sample is *correct* provided the path the sample takes through the tree starting from the root node ends at the leaf corresponding to the correct label.

3 Integer Programming Formulation

In this section we provide an integer programming formulation of the above problem and discuss some tricks to make it more efficient for the MILP solver.

3.1 The basic formulation

We begin with some notation. Let the set of all samples be indexed by $I = \{1, 2, \dots, N\}$ and let $I_+ \subset I$ denote the indices i of samples such that $y_i = 1$, denote $I_- = I \setminus I_+$. Let the set of groups be indexed by $G = \{1, 2, \dots, |G|\}$ and the set of features be indexed by $J = \{1, 2, \dots, d\}$, in addition let $g(j) \in G$ denote the group that contains feature $j \in J$ and let $J(g)$ denote the set of features that are contained in group g . We assume that the topology of the decision tree is given and let the set of the decision nodes be indexed by $K = \{1, 2, \dots, |K|\}$ and the set of leaves be indexed by $B = \{1, 2, \dots, |B|\}$. We denote the indices of leaves with positive labels by $B_+ \subset B$ and the indices of leaves with negative labels by $B_- = B \setminus B_+$. For convenience, we let B_+ contain even indices, and B_- contain the odd ones.

We will now describe our key decision variables and the constraints on these variables. We use binary variables $v_g^k \in \{0, 1\}$ for $g \in G$ and $k \in K$ to denote if group g is selected for branching at node k . As discussed earlier in Section 2, exactly one group is selected for branching at a node and consequently we have the following set of constraints:

$$\sum_{g \in G} v_g^k = 1 \quad (1)$$

for all $k \in K$ in the formulation.

The second set of variables $z_j^k \in \{0, 1\}$ denote if feature $j \in J$ is one of the selected features used for branching at node $k \in K$. Clearly, a feature $j \in J$ can be selected only if the group containing it, namely $g(j)$, is selected at that node. Therefore we have the following set of constraints:

$$z_j^k \leq v_{g(j)}^k \quad \forall j \in J, \quad (2)$$

in the formulation. Let

$$S = \{z \in \{0, 1\}^{|K| \times d}, v \in \{0, 1\}^{|K| \times |G|} : \\ (v, z) \text{ satisfy inequalities (1) and (2)}\},$$

and note that for any $(v, z) \in S$ one can construct a corresponding decision tree in a unique way and vice versa. We next relate these variables (and therefore the corresponding decision tree) to the samples.

We use variables $c_b^i \in \{0, 1\}$ for $b \in B$ and $i \in I$ to denote if sample i is routed to leaf node b . This means that $c_b^i = 1$ only when the sample i exactly follows the right or left branches of the decision nodes that lead to leaf node b . With this in mind, we define the following expression

$$L(i, k) = \sum_{j \in J} a_j^i z_j^k \quad (3)$$

and make the following observation:

Proposition 3.1. *Let $(z, v) \in S$, then, $L(i, k) \in \{0, 1\}$ for all $i \in I$ and $k \in K$. Furthermore, $L(i, k) = 1$ if and only if there exists some $j \in J$ such that $a_j^i = 1$ and $z_j^k = 1$.*

Consequently, the expression $L(i, k)$ denotes if sample $i \in I$ branches left at node $k \in K$. Similarly, we also define the expression

$$R(i, k) = 1 - L(i, k) \quad (4)$$

to indicate that sample i branches right at node k .

To complete the model, we next relate these expressions to the c_b^i variables which denote if sample $i \in I$ is routed to leaf node $b \in B$. As the topology of the tree is fixed, there is a unique path leading to each leaf node $b \in B$ from the root of the tree. This path visits a subset of the nodes $K(b) \subset K$ and for each $k \in K(b)$ either the left branch or the right branch is followed. Let $K^L(b) \subseteq K(b)$ denote the nodes where the left branch is followed and let $K^R(b) = K(b) \setminus K^L(b)$ denote the nodes where the right branch is followed. Sample i is routed to b only if it satisfies all the conditions at the nodes leading to that leaf node. Consequently, we define:

$$c_b^i \leq L(i, k) \quad \text{for all } k \in K^L(b) \quad (5)$$

$$c_b^i \leq R(i, k) \quad \text{for all } k \in K^R(b) \quad (6)$$

for all $i \in I$ and $b \in B$. Combining these with

$$\sum_{b \in B} c_b^i = 1 \quad (7)$$

for all $i \in I$ gives a complete formulation. Let

$$Q(z, v) = \{c \in \{0, 1\}^{N \times |B|} : \text{such that (5)-(7) hold}\}.$$

We next show that combining the constraints defining sets S and $Q(z, v)$ leads to a correct formulation.

Proposition 3.2. *Let $(z, v) \in S$, and $c \in Q(z, v)$. Then, $c_b^i \in \{0, 1\}$ for all $i \in I$ and $b \in B$. In addition, if $c_b^i = 1$ then sample i belongs to leaf node b .*

We therefore have the following integer programming (IP) formulation:

$$\max \quad \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i \quad (8a)$$

$$\text{s. t.} \quad (z, v) \in S \quad (8b)$$

$$c \in Q(z, v) \quad (8c)$$

where C in the objective (8a) is a constant weight chosen to deal with class imbalance. For instance, if a training set has twice as many good examples as bad examples, we set $C = 2$, so that every correct classification of a bad data point is equal to two correct classifications of good data points.

3.2 Computationally Tractability

While (8) is a correct formulation, it can be improved to enhance computational performance.

3.2.1 Deleting unnecessary variables

Notice that the objective function (8a) uses variables c_b^i only if $i \in I_+$ and $b \in B_+$, or $i \in I_-$ and $b \in B_-$. Consequently, the remaining c_b^i variables can be projected out from the formulation without changing the value of the optimal solution. We therefore, only define c_b^i variables for

$$\{(i, b) : i \in I_+, b \in B_+, \text{ or, } i \in I_-, b \in B_-\} \quad (9)$$

and write constraints (5) and (6) only for these variables. In addition, we delete equation (7).

Also note that the objective function (8a) is maximizing a (weighted) sum of c_b^i variables. Therefore, if we replace the integrality constraints $c_b^i \in \{0, 1\}$ with simple bound constraints $1 \geq c_b^i \geq 0$, the optimal solution would still satisfy $c_b^i \in \{0, 1\}$. Consequently, we do not require c_b^i to be integral in the formulation.

3.2.2 Breaking symmetry: Anchor features

If the variables of an integer program can be permuted without changing the structure of the problem, the integer program is called *symmetric*. This poses a problem for IP solvers (such as IBM ILOG Cplex) as the search space increases exponentially, see Margot (2009). The formulation (8) falls into this category as there are multiple alternate solutions that represent the same decision tree. In particular, if the branching condition is reversed at a node other than a leaf node, and, at the same time, the sub-trees associated with the right and left branches of the node are switched, one obtains an alternate solution corresponding to the same decision tree. To avoid this, we designate one particular feature $j(g) \in J(g)$ of each group $g \in G$ to be the *anchor feature* of that group and enforce that if a group is selected for branching at a node, samples with the anchor feature follow the left branch. More precisely, we add the following equations to the formulation:

$$z_{j(g)}^k = v_{j(g)}^k \quad (10)$$

for all $g \in G$, and all $k \in K$ that is not adjacent to a leaf node. While equations (10) lead to better computational performance, they do not rule out any decision trees from the feasible set of solutions.

3.2.3 Relaxing some binary variables

The computational difficulty of an IP typically increases with the number of integer variables in the formulation and therefore it is desirable to declare as few variables as possible as integral. Consider a feature selection variable z_j^k where $j \in J$ and $k \in K$ is a node that is not adjacent to a leaf node. We will next argue that these variables only take values $\{0, 1\}$ in an optimal solution even when they are not explicitly required to be integral.

Proposition 3.3. *Every extreme point solution to (8) is integral even if the variables z_j^k , where $j \in J$ and $k \in K$ is a node that is not adjacent to a leaf node, are not declared integral in the definition of S .*

3.2.4 Strengthening the model

Now consider the inequalities (5)

$$c_b^i \leq L(i, k) \quad \text{for all } k \in K^L(b) \quad (11)$$

for $i \in I$ and $b \in B$ and remember that $\sum_{b \in B} c_b^i = 1$ due to equation (7) for $i \in I$. Consequently, for any $i \in I$, if $L(i, k) = 0$ then all $c_b^i = 0$ if $k \in K^L(b)$. In addition, if $L(i, k) = 1$ then at most one $c_b^i = 1$ if $k \in K^L(b)$. Therefore,

$$\sum_{b \in B: k \in K^L(b)} c_b^i \leq L(i, k) \quad (12)$$

is a valid inequality for all $i \in I$. While this inequality is satisfied by all integral solutions to the set $Q(z, v)$, it is violated by solutions to its continuous relaxation. We replace inequalities the inequalities (5) in the formulation with (12) to obtain a tighter formulation. We do the same for $R(i, k)$.

4 Computational Results

In all our experiments, we first defined a tree topology as depth 2, depth 2.5 and depth 3. Each dataset was preprocessed to have the binary form assumed by the formulation, with identified groups of binary variables. Each dataset/tree topology pair results in a MILP formulation, which we implemented in Python 2.7 and then solved with IBM ILOG CPLEX 12.6.1 on a machine with an 8-core 2.5 GHz Intel Core i7 processor. Throughout this section, we refer to our method as ODT (Optimal Decision Trees).

In our first set of experiments, we sketch the accuracy of our DTs as a function of the number of training examples and the depth of the tree. We use the well known Wisconsin *breast cancer* dataset available from the UCI repository [8]. After preprocessing the data to remove missing values, the dataset had 695 examples and 90 features, which could be grouped into 9 groups. We consider the 3 tree sizes mentioned above and 5 training sample sizes, $\{100, 200, 300, 400, 500\}$. For each of the 3×5 combinations of topology and sample size, we randomly partitioned the 695 examples into a training set of the given size and a holdout testing set of the remaining examples. Once the split was generated, we counted the number of positive instances p (corresponding to benign tumors in this dataset) and the number of negative instances n (corresponding to malignant tumors) in the training set. We then set $C = p/n$ in our model to crudely account for class imbalance. The average training and testing accuracy of the classifiers learned by running this experiment 10 times is presented in the bar graphs in Figure 3.

In this experiment, all instances were solved to global optimality. We observe that there are natural and expected tradeoffs in tree topology and fraction of the dataset used for training. In particular, while a deeper tree and fewer examples lead to better training accuracy, deeper trees (depth 3) tend to overfit the training data and perform worse in testing accuracy.

In Table 1, we list several datasets from the UCI Machine Learning Repository [8] of moderate size that contain integer or categorical features, making them appropriate for our setting. By “math scores”, we are referring to the Student Alcohol Consumption dataset, for predicting Portuguese secondary school students’ failure of a mathematics course [5]. For all datasets, we preprocessed the

Figure 3: Average training (yellow) and testing (blue) accuracy of our classifier on the breast cancer dataset, varying training sample size and tree topology (depth). Left bar is Depth 2, center bar is Depth 2.5, right bar is Depth 3.

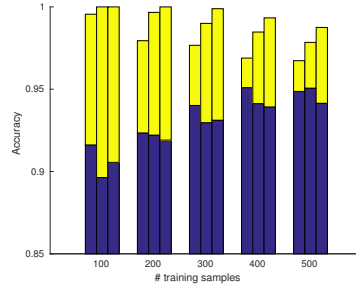


Table 1: Properties of datasets used

	Samples	Pos/Neg	Features	Groups
adult	1605	395/1210	122	14
breast can.	695	454/241	90	9
letter	20000	734/19246	320	20
math scores	395	265/130	109	31
mushrooms	8124	4208/3916	111	20

data, deleting either features or samples (whichever resulted in fewer deletions) whenever missing entries were encountered.

Using our MILP model, we generated trees of the three sizes for each of the data set. A 4 hour time limit was given to solve each instance. In the case when the time limit was reached, a feasible tree was always obtained, but without a certificate of global optimality. Nonetheless, these feasible but possibly suboptimal trees achieve good accuracy as evidenced in Table 2. For each dataset, we chose a number of training examples (given in the parenthesis) intended both 1) to give a reasonable balance between the size of training and testing sets, and 2) provide a tractable model in terms of numbers of variables and constraints. We compare against the implementation of CART [3] and random forest classifiers [4] in scikit-learn [9]. For each classifier and dataset, we used a random partition of the data of the size shown in the table, and ran it 10 times, recording the average training and testing accuracies.

We do not compare the running times since it is clear that CART and RF classifiers can be obtained very quickly. The aim of our computational comparison is to show that our trees can be obtained in a reasonable amount of time and provide much better accuracy than traditional DTs. By simply comparing the training and testing accuracy, we see that the ODT method produces classifiers with good accuracy that are not prone to overfitting. On the other hand, CART and RF overfit on all data sets aside from *letter*, which is highly imbalanced - hence the accuracy

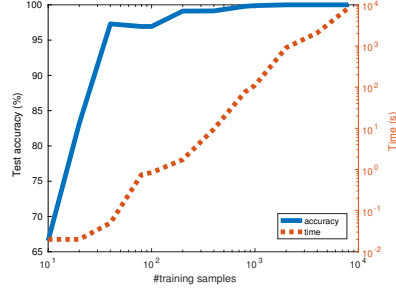
Table 2: Comparing ODT to CART and random forests. Training Accuracy/Testing Accuracy

	CART	RF	ODT, Depth 3	ODT, Depth 2.5	ODT, Depth 2
adult (1000)	85.54/61.34	96.13/71.31	77.80/74.12	74.29/72.76	75.21/71.75
breast cancer (500)	74.28/52.62	85.14/56.00	98.75/94.14	97.84/95.07	96.73/94.86
letter (1000)	97.44/93.46	99.10/96.24	97.69/94.88	93.59/91.54	86.90/86.69
math scores (250)	83.40/56.34	99.00/59.86	94.00/88.93	92.70/90.52	92.60/90.69
mushrooms (1000)	92.96/47.46	99.16/46.57	100.00/99.89	99.92/99.79	99.35/99.42

Table 3: Average Testing Sensitivity/Specificity over 10 trials for CART, random forests, and ODT

	CART	RF	ODT, Depth 3	ODT, Depth 2.5	ODT, Depth 2
adult (1000)	28.85/79.73	5.13/94.88	86.02/70.54	89.12/68.22	87.73/67.00
breast cancer (500)	64.90/40.91	58.94/45.45	97.35/89.44	95.99/93.70	95.66/95.71
letter (1000)	3.69/97.13	0.01/99.97	63.31/94.96	80.61/91.96	89.68/86.58
math scores (250)	54.08/42.55	74.49/29.79	90.33/83.64	92.28/86.21	90.58/90.84
mushrooms (1000)	90.69/9.10	96.95/3.70	100.00/99.77	100.00/99.55	100.00/98.7

Figure 4: Comparing test accuracy and time necessary to construct classifier as a function of number of training samples used to train classifier



measure is misleading. Even though we used the settings in CART and RF algorithms that are intended to account for class imbalance, the classifiers learned by these methods did not perform well in terms of sensitivity and specificity when applied to the holdout testing set. In contrast, the classifiers learned by our method tend to balance sensitivity and specificity well on both training and testing data. See Table 3 for results on the sensitivity and specificity of classifiers on the testing data. We note that increasing the size of the training data sets for CART and RF did not result in a significant improvement, hence the training set sizes were not a defining factor in the comparatively poor accuracy of these algorithms. We also note that on the *math scores* data set the ODTs outperformed all of the classification results reported in [5].

We tested the empirical complexity and the accuracy of our models with respect to the number of training instances. In Figure 4, using the *mushrooms* dataset, we see that the solution time increases roughly quadratically with the number of instances. In all these runs, a depth 3 tree was trained using a random sample of the specified size. At the same time we observe that a perfect depth 3 classifier can be obtained with a training set of 1000 samples; thus, there is no practical reason to exceed this threshold. Hence, in our setting, learning is possible with small or moderate data sets.

Finally, we report in Table 4 the minimum and maximum times, over 10 trials, required by CPLEX to solve the model for different datasets and tree topologies. Since a time limit of 4 hours was imposed, an *X* in Table 4 denotes that this upper bound was hit and CPLEX returned the feasible solution of the highest objective value that was found. We remark from a combinatorial perspective that even with the proposed shallow depths, the task of finding an ODT is nontrivial; indeed, consider enumerating all depth 2 decision trees for the breast cancer dataset. There are 9 groups with 10 features per group; thus, at the root node, there are $9 \cdot 2^9$ possible branching decisions (due to the anchor features), and at the two non-root decision nodes, there are $9 \cdot 2^{10}$ possible branching decisions, for a total of $9^3 \cdot 2^{29}$ possible trees, approximately 391 billion. For a depth 3 tree, there are $9^7 \cdot 2^{67}$ possible trees, approximately 7×10^{26} .

Table 4: [Minimum,Maximum] time (in seconds) required to solve the model over 10 trials.

	Depth 2	Depth 2.5	Depth 3
adult (1000)	[49,55]	[776,4207]	[X,X]
breast can. (500)	[8,11]	[175,659]	[4881,7976]
letter (1000)	[112,128]	[11651,X]	[X,X]
math scores (250)	[15,26]	[1199,4120]	[X,X]
mushrooms (1000)	[17,62]	[115,928]	[171,401]

We note that although in many depth 3 instances, the solver reached the time limit, we have seen that the quality of the feasible solution returned is still satisfactory for learning purposes. Moreover, a direction of future work involves generating random forests of short (e.g. depth 2 and depth 2.5) ODTs, which we have seen often outperform single CARTs as predictors, leading us to believe that random forests of ODTs have the potential to outperform random forests of CARTs.

5 Extensions

5.1 Maximize Sensitivity/Specificity

We next describe how to change the model (8) to build a decision tree that maximizes sensitivity (the true positive rate, or TPR) while guaranteeing a certain level of specificity. For example if we would like to train a classifier with a guaranteed specificity of 0.95 then we add the constraint to the model (8)

$$\sum_{i \in I_-} \sum_{b \in B_-} c_b^i \geq \lceil (1 - 0.95)|I_-| \rceil \quad (13)$$

and change the objective function (8a) to

$$\sum_{i \in I_+} \sum_{b \in B_+} c_b^i. \quad (14)$$

Likewise, we can produce a model that maximizes specificity while guaranteeing a certain level of specificity by switching the expressions in the constraint (13) and objective (14).

5.2 Handling Real Valued Features

The IP model (8) can be extended to handle real-valued features in addition to categorical features. In this case branching decisions are *either* made based on a group of categorical features, *or*, based on the real-valued features. If real-valued features are chosen for branching, then the samples are divided according to a linear classifier optimally chosen by the extended model.

The extended model has a new binary variable $v_{cg}^k \in \{1, 0\}$ to denote if the real-valued features are chosen for branching at node $k \in K$. In addition, we also define variables $w^k \in \mathbb{R}^n$ and $u^k \in \mathbb{R}$ for each $k \in K$ to denote the linear classifier used for branching. Finally we define a new binary variable $H(i, k) \in \{0, 1\}$ for $i \in I, k \in K$ to denote if sample i satisfies the conditions of the linear classifier at node k .

Using these variables we can write a set of constraints relating each $H(i, k)$ to the associated linear classifier. A second set of constraints can be used to combine $H(i, k)$ and v_{cg}^k variables to determine if a sample should go left at node k .

We note that this approach can be further extended to deal with multiple groups of real-valued features. We are currently in the process of experimenting with these extensions, however, the purpose of this paper is to exploit the structure of the categorical variables, hence we do not include further details here (see the supplementary material for a more details on the formulation).

We next extend the IP model (8) to deal with real-valued features. Many datasets from real applications include such features together with categorical features. We assume that the branching decisions in the decision tree are *either* made based on a group of categorical features, *or*, based on the real-valued features. In other words, we now have a new group of features consisting of the real-valued features. If this group is chosen for branching, then we divide the samples according to a linear classifier optimally chosen by the extended model.

Given a sample $i \in I$, let the last n features in a^i be real-valued and denote them with $a_{cg}^i \in \mathbb{R}^n$. The extended model has a binary variable $v_{cg}^k \in \{1, 0\}$ to denote if the real-valued features are chosen for branching at node $k \in K$. This variable is added to equation 1 to make sure that only one group is chosen for branching at any node. We also define variables $w^k \in \mathbb{R}^n$ and $u^k \in \mathbb{R}$ for each $k \in K$ to denote the linear classifier used for branching. Sample $i \in I$ moves left at node $k \in K$ if $\langle w^k, a_{cg}^i \rangle + u^k > 0$ and moves right if $\langle w^k, a_{cg}^i \rangle + u^k < 0$.

We now define a new binary variable $H(i, k) \in \{0, 1\}$ for $i \in I, k \in K$ to denote if sample i satisfies the conditions of the linear classifier at node k to go left and write the following constraints for $k \in K, i \in I$:

$$\begin{aligned} 1 - H(i, k) &\geq -(\langle w^k, a_{cg}^i \rangle + u^k) + \epsilon \\ H(i, k) &\geq \langle w^k, a_{cg}^i \rangle + u^k + \epsilon \end{aligned} \quad (15)$$

where $\epsilon > 0$ is a small perturbation. Clearly $H(i, k) = 1$ if and only if $\langle w^k, a_{cg}^i \rangle + u^k > 0$. Moreover, these constraints force $\langle w^k, a_{cg}^i \rangle + u^k \in (-\epsilon, \epsilon)$ to be infeasible.

We finally define the binary variables, $\Lambda(i, k) \in \{0, 1\}$ to denote if the following two conditions are satisfied simultaneously: (1) the continuous group is selected for branching at node $k \in K$, and, (2) sample $i \in I$ satisfies the conditions of the linear classifier to go left. The following inequalities are known as the McCormick inequalities and they are commonly used to represent the product of two binary variables. For all $k \in K, i \in I$ we write:

$$\begin{aligned} \Lambda(i, k) &\geq v_{cg}^k + H(i, k) - 1 \\ \Lambda(i, k) &\leq v_{cg}^k \\ \Lambda(i, k) &\leq H(i, k) \\ \Lambda(i, k) &\geq 0 \end{aligned} \quad (16)$$

Clearly the variable $\Lambda(i, k)$ has the desired meaning that sample i should go left at node k due to the linear classifier. With this setup, we can easily extend (8) by redefining $L(i, k)$ from (3) as

$$L(i, k) = \sum_{j \in J \setminus \{j: j \in cg\}} a_j^i z_j^k + \Lambda(i, k). \quad (17)$$

to obtain a correct model that can handle real-valued features.

We also note that this approach can be extended to deal with multiple groups of real-valued features. In this case, one needs to define the variables and constraints mentioned above for all $cg \in \{cg_1, cg_2, \dots\}$ where each cg_l denotes a group of real-valued features.

References

- [1] T. Achterberg and R. Wunderling. Mixed integer programming: Analyzing 12 years of progress. In Juenger et al, editor, *Facets of Combinatorial Optimization*. Springer, 2009.
- [2] R. Bixby. A brief history of linear and mixed-integer programming computation. *DOCUMENTA MATHEMATICA*, pages 107–121, 2010.
- [3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
- [4] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001.
- [5] P. Cortez and A. Silva. Using data mining to predict secondary school student performance. In A. Brito and J. Teixeira, editors, *Proceedings of 5th FUTURE BUSINESS TECHNOLOGY Conference*, pages 5–12. FUBUTEC 2008, 2008.
- [6] S. B. Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.
- [7] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.
- [8] M. Lichman. UCI machine learning repository, 2013.
- [9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [10] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

Supplementary material: Proofs

Proof of Proposition 3.1

Proof. For any $(z, v) \in S$ and $k \in K$, exactly one of the v_g^k variables, say $v_{g'}^k$, takes value 1 and $v_g^k = 0$ for all $g \neq g'$. Therefore, $z_j^k = 0$ for all $j \notin J(g)$. Consequently, the first claim follows for all $i \in I$ as $L(i, k) = \sum_{j \in J} a_j^i z_j^k = \sum_{j \in J(g')} a_j^i z_j^k = z_{j_i}^k \in \{0, 1\}$ where $j_i \in J(g')$ is the index of the unique feature for which $a_{j_i}^i = 1$. Consequently, $L(i, k) = 1$ if and only if $z_{j_i}^k = 1$ which proves our claim. \square

Proof of Proposition 3.2

Proof. Given $(z, v) \in S$, and $i \in I$ assume that sample i should be routed to leaf node b' (according to the value of its features and the rules on the given decision tree defined by (z, v)). For all other leaves $b \in B \setminus \{b'\}$, the sample, must have either $L(i, k) = 0$ for some $k \in K^L(b)$ or $R(i, k) = 0$ for some $k \in K^R(b)$. Consequently, $c_b^i = 0$ for all $b \neq b'$. Equation (7) then implies that $c_{b'}^i = 1$ and

therefore $c_b^i \in \{0, 1\}$ for all $b \in B$. Conversely, if $c_{b'}^i = 1$ for some $b' \in B$, then $L(i, k) = 1$ for all $k \in K^L(b)$ and $R(i, k) = 1$ for all $k \in K^R(b)$. \square

Proof of Proposition 3.3

Proof. Consider an extreme point solution (v, z, c) to the relaxed formulation and assume that the claim does not hold. Then $z_{j'}^k$ is fractional for some $j \in J$ and $k \in K$ where node k is adjacent to leaf nodes $b_+(k)$ and $b_-(k)$. Due to inequality (2), $v_{g(j')}^k = 1$ as $z_{j'}^k > 0$. Let $I_0 \subset I$ be the collection of samples that satisfy the conditions to follow the path leading to node k . Clearly, $c_{b_+(k)}^i$ and $c_{b_-(k)}^i$ variables associated with all $i \in I \setminus I_0$ are zero in the solution.

We will now construct two solutions (v, \bar{z}, \bar{c}) and (v, \hat{z}, \hat{c}) by replacing $z_{j'}^k$ with $(z_{j'}^k + \epsilon)$ and $(z_{j'}^k - \epsilon)$, respectively, for some small $\epsilon > 0$, and modifying the c variables associated with samples $i \in I_0$ accordingly. We will then argue that $(v, z, c) = 1/2(v, \bar{z}, \bar{c}) + 1/2(v, \hat{z}, \hat{c})$ and therefore is not an extreme point solution. Remember that in the formulation we have $c_{b_+(k)}^i$ variables associated with $i \in I_+$ and $c_{b_-(k)}^i$ variables associated with $i \in I_-$ only. In addition,

$$c_{b_+(k)}^i \leq R(i, k) = 1 - \sum_{j \in J} a_j^i z_j^k \quad \forall i \in I_0 \cap I_+ \quad (18)$$

$$c_{b_-(k)}^i \leq L(i, k) = \sum_{j \in J} a_j^i z_j^k \quad \forall i \in I_0 \cap I_- \quad (19)$$

Now consider $i \in I_0$. If $a_{j'}^i = 0$, then we do not change the value of the associated c variable in the perturbed solutions (v, \bar{z}, \bar{c}) and (v, \hat{z}, \hat{c}) . If $a_{j'}^i = 1$, we consider two cases. If $i \in I_+$ and $c_{b_+(k)}^i = 0$, or, $i \in I_-$ and $c_{b_-(k)}^i = 0$, then we again do not change the value of the associated c variable in the perturbed solutions. Finally, for the remaining $i \in I_+$ we set $\bar{c}_{b_+(k)}^i = c_{b_+(k)}^i + \epsilon$ and $\hat{c}_{b_+(k)}^i = c_{b_+(k)}^i - \epsilon$. Similarly, for the remaining $i \in I_-$ we set $\bar{c}_{b_-(k)}^i = c_{b_-(k)}^i - \epsilon$ and $\hat{c}_{b_-(k)}^i = c_{b_-(k)}^i + \epsilon$.

It is easy to check that (v, \bar{z}, \bar{c}) and (v, \hat{z}, \hat{c}) are feasible solutions and contain (v, z, c) in their convex hull. \square

Details on extending the formulation to include the continuous variables

Given a sample $i \in I$, let the last n features in a^i be real-valued and denote them with $a_{cg}^i \in \mathbb{R}^n$. The extended model has a binary variable $v_{cg}^k \in \{1, 0\}$ to denote if the real-valued features are chosen for branching at node $k \in K$. This variable is added to equation 1 to make sure that only one group is chosen for branching at any node. We also define variables $w^k \in \mathbb{R}^n$ and $u^k \in \mathbb{R}$ for each $k \in K$ to denote the linear classifier used for branching. Sample $i \in I$ moves left at node $k \in K$ if $\langle w^k, a_{cg}^i \rangle + u^k > 0$ and moves right if $\langle w^k, a_{cg}^i \rangle + u^k < 0$.

We now define a new binary variable $H(i, k) \in \{0, 1\}$ for $i \in I, k \in K$ to denote if sample i satisfies the conditions of the linear classifier at node k to go left and write the following constraints for $k \in K, i \in I$:

$$\begin{aligned} 1 - H(i, k) &\geq -(\langle w^k, a_{cg}^i \rangle + u^k) + \epsilon \\ H(i, k) &\geq \langle w^k, a_{cg}^i \rangle + u^k + \epsilon \end{aligned} \quad (20)$$

where $\epsilon > 0$ is a small perturbation. Clearly $H(i, k) = 1$ if and only if $\langle w^k, a_{cg}^i \rangle + u^k > 0$. Moreover, these constraints force $\langle w^k, a_{cg}^i \rangle + u^k \in (-\epsilon, \epsilon)$ to be infeasible.

We finally define the binary variables, $\Lambda(i, k) \in \{0, 1\}$ to denote if the following two conditions are satisfied simultaneously: (1) the continuous group is selected for branching at node $k \in K$, and, (2) sample $i \in I$ satisfies the conditions of the linear classifier to go left. The following inequalities are known as the McCormick inequalities and they are commonly used to represent the product of two binary variables. For all $k \in K, i \in I$ we write:

$$\begin{aligned}\Lambda(i, k) &\geq v_{cg}^k + H(i, k) - 1 \\ \Lambda(i, k) &\leq v_{cg}^k \\ \Lambda(i, k) &\leq H(i, k) \\ \Lambda(i, k) &\geq 0\end{aligned}\tag{21}$$

Clearly the variable $\Lambda(i, k)$ has the desired meaning that sample i should go left at node k due to the linear classifier. With this setup, we can easily extend (8) by redefining $L(i, k)$ from (3) as

$$L(i, k) = \sum_{j \in J \setminus \{j: j \in cg\}} a_j^i z_j^k + \Lambda(i, k).\tag{22}$$

to obtain a correct model that can handle real-valued features.

We also note that this approach can be extended to deal with multiple groups of real-valued features. In this case, one needs to define the variables and constraints mentioned above for all $cg \in \{cg_1, cg_2, \dots\}$ where each cg_l denotes a group of real-valued features.